Practical Cryptanalysis of ISO/IEC 9796-2 and EMV Signatures[⋆]

Jean-Sébastien Coron[1], David Naccache[2], Mehdi Tibouchi[2], and Ralf-Philipp Weinmann[1]

[1] Université du Luxembourg
6, rue Richard Coudenhove-Kalergi
L-1359 Luxembourg, Luxembourg
{jean-sebastien.coron, ralf-philipp.weinmann}@uni.lu
[2] École normale supérieure
Département d'informatique, Groupe de Cryptographie
45, rue d'Ulm, F-75230 Paris CEDEX 05, France
{david.naccache, mehdi.tibouchi}@ens.fr

Abstract. In 1999, Coron, Naccache and Stern discovered an existential signature forgery for two popular RSA signature standards, ISO/IEC 9796-1 and 2. Following this attack ISO/IEC 9796-1 was withdrawn. ISO/IEC 9796-2 was amended by increasing the message digest to at least 160 bits. Attacking this amended version required at least 2^{61} operations.

In this paper, we exhibit algorithmic refinements allowing to attack the *amended* (currently valid) version of ISO/IEC 9796-2 for all modulus sizes. A practical forgery was computed in only two days using 19 servers on the Amazon EC2 grid for a total cost of $\simeq$ US\$800. The forgery was implemented for $e = 2$ but attacking odd exponents will not take longer. The forgery was computed for the RSA-2048 challenge modulus, whose factorization is still unknown.

The new attack blends several theoretical tools. These do not change the asymptotic complexity of Coron *et al.*'s technique but significantly accelerate it for parameter values previously considered beyond reach.

While less efficient (US\$45,000), the acceleration also extends to EMV signatures. EMV is an ISO/IEC 9796-2-compliant format with extra redundancy. Luckily, this attack does not threaten any of the 730 million EMV payment cards in circulation for *operational* reasons.

Costs are per modulus: after a first forgery for a given modulus, obtaining more forgeries is virtually immediate.

keywords: digital signatures, forgery, RSA, public-key cryptanalysis, ISO/IEC 9796-2, EMV.

1 Introduction

RSA [49] is certainly the most popular public-key cryptosystem. A chosen-ciphertext attack against RSA textbook encryption was described by Desmedt and Odlyzko in [21]. In RSA textbook encryption, a message m is simply encrypted as:

$$c = m^e \mod N$$

where N is the RSA modulus and e is the public exponent.

As noted in [42], Desmedt and Odlyzko's attack also applies to RSA signatures:

$$\sigma = \mu(m)^d \mod N$$

where $\mu(m)$ is an encoding function and d the private exponent. Desmedt and Odlyzko's attack only applies if the encoding function $\mu(m)$ produces integers much smaller than N. In which case, one obtains an existential forgery under a chosen-message attack. In this attack the opponent can ask for signatures of any messages of his choosing before computing, by his own means, the signature of a (possibly meaningless) message which was never signed by the legitimate owner of d.

As of today, two encoding function species co-exist:

[⋆] An extended abstract of this paper will appear at CRYPTO 2009. This is the full version.

1. *Ad-hoc encodings* are "handcrafted" to thwart certain classes of attacks. While still in use, *ad-hoc* encodings are currently being phased-out. PKCS#1 v1.5 [32], ISO/IEC 9796-1 [28] and ISO/IEC 9796-2 [29, 30] are typical *ad-hoc* encoding examples.
2. *Provably secure encodings* are designed to make cryptanalysis equivalent to inverting RSA (possibly under additional assumptions such as the Random Oracle Model [2]). OAEP [3] (for encryption) and PSS [4] (for signature) are typical provably secure encoding examples.

For *ad-hoc* encodings, there is no guarantee that forging signatures is as hard as inverting RSA. And as a matter of fact, many such encodings were found to be weaker than the RSA problem. We refer the reader to [11, 15, 14, 31, 18, 25] for a few characteristic examples. It is thus a practitioner's rule of thumb to use provably secure encodings whenever possible. Nonetheless, *ad-hoc* encodings continue to populate hundreds of millions of commercial products (*e.g.* EMV cards) for a variety of practical reasons. A periodic re-evaluation of such encodings is hence necessary.

ISO/IEC 9796-2 is a specific μ-function standardized by ISO [29]. In [19], Coron, Naccache and Stern discovered an attack against ISO/IEC 9796-2. Their attack is an adaptation of Desmedt and Odlyzko's cryptanalysis which could not be applied directly since in ISO/IEC 9796-2, the encoding $\mu(m)$ is almost as large as N. ISO/IEC 9796-2 can be used with hash-functions of diverse digest-sizes k_h. Coron *et al.* estimated that attacking $k_h = 128$ and $k_h = 160$ will require (respectively) 2^{54} and 2^{61} operations. After Coron *et al.*'s publication ISO/IEC 9796-2 was amended and the current official requirement (see [30]) is $k_h \geq 160$. It was shown in [16] that ISO/IEC 9796-2 can be proven secure (in the random oracle model) for $e = 2$ and if the digest size k_h is a least 2/3 the size of the modulus.

In this paper, we describe an improved attack against the currently valid (amended) version of ISO/IEC 9796-2, that is for $k_h = 160$. The new attack applies to EMV signatures as well. EMV is an ISO/IEC 9796-2-compliant format with extra redundancy. The attack is a Coron *et al.* forgery with new refinements: better message choice, Bernstein's smoothness detection algorithm (instead of trial division), large prime variant and optimized exhaustive search.

Using these refinements, a forgery was computed in only two days, using a few dozens of servers on the Amazon EC2 grid, for a total cost of US\$800. The forgery was implemented for $e = 2$ but attacking odd exponents will not take longer. We estimate that under similar conditions an EMV signature forgery would cost US\$45,000. Note that all costs are per modulus. After computing a first forgery for a given N, additional forgeries come at a negligible cost.

2 The ISO/IEC 9796-2 Standard

ISO/IEC 9796-2 is an encoding standard allowing partial or total message recovery [29, 30]. Here we consider only partial message recovery. As we have already mentioned, ISO/IEC 9796-2 can be used with hash-functions HASH(m) of diverse digest-sizes k_h. For the sake of simplicity we assume that k_h, the size of m and the size of N (denoted k) are all multiples of 8;[1] this is also the case in the EMV specifications.

The ISO/IEC 9796-2 encoding function is:

$$\mu(m) = \texttt{6A}_{16} \| m[1] \| \text{HASH}(m) \| \texttt{BC}_{16}$$

[1] As will be clarified later, if we drop the assumption that $k \equiv 0 \mod 8$, the attack actually becomes faster. Indeed, if m consists of an odd number of 4-bit nibbles, the encoding function becomes $\mu(m) = \texttt{7}_{16} \| m[1] \| \text{HASH}(m) \| \texttt{BC}_{16}$, with only 12 fixed bits instead of 16, which in turn makes the involved integers about 4 bits shorter. *e.g.* the RSA-2048 $\{a, b\}$-pair (*cf.* section 5.3) becomes $\{45, 28\}$ instead of $\{625, 332\}$.

where the message $m = m[1]\|m[2]$ is split in two: $m[1]$ consists of the $k - k_h - 16$ leftmost bits of m and $m[2]$ represents all the remaining bits of m. The size of $\mu(m)$ is therefore always $k - 1$ bits.

The original version of the standard recommended $128 \leq k_h \leq 160$ for partial message recovery (see [29], §5, note 4). The new version of ISO/IEC 9796-2 [30] requires $k_h \geq 160$. The EMV specifications also use $k_h = 160$.

3 Desmedt-Odlyzko's Attack

In Desmedt and Odlyzko's attack [42] (existential forgery under a chosen-message attack), the forger asks for the signature of messages of his choice before computing, by his own means, the signature of a (possibly meaningless) message that was never signed by the legitimate owner of d. In the case of Rabin-Williams signatures (see Appendix A), it may even happen that the attacker factors N; *i.e.* a total break.

The attack only applies if $\mu(m)$ is much smaller than N and works as follows:

1. Select a bound B and let $\mathfrak{P} = \{p_1, \ldots, p_\ell\}$ be the list of all primes smaller than B.
2. Find at least $\ell + 1$ messages m_i such that each $\mu(m_i)$ is a product of primes in $\mathfrak{P}$.
3. Express one $\mu(m_j)$ as a multiplicative combination of the other $\mu(m_i)$, by solving a linear system given by the exponent vectors of the $\mu(m_i)$ with respect to the primes in $\mathfrak{P}$.
4. Ask for the signatures of the m_i for $i \neq j$ and forge the signature of m_j.

In the following we assume that e is prime; this includes $e = 2$. We let τ be the number of messages m_i obtained at step 2. We say that an integer is B-smooth if all its prime factors are smaller than B. The integers $\mu(m_i)$ obtained at step 2 are therefore B-smooth and we can write for all messages m_i, $1 \leq i \leq \tau$:

$$\mu(m_i) = \prod_{j=1}^{\ell} p_j^{v_{i,j}} \tag{1}$$

To each $\mu(m_i)$ we associate the ℓ-dimensional vector of the exponents modulo e:

$$\boldsymbol{V}_i = (v_{i,1} \bmod e, \ldots, v_{i,\ell} \bmod e)$$

Since e is prime, the set of all ℓ-dimensional vectors modulo e forms a linear space of dimension ℓ. Therefore, if $\tau \geq \ell + 1$, one can express one vector, say $\boldsymbol{V}_\tau$, as a linear combination of the others modulo e, using Gaussian elimination, which gives for all $1 \leq j \leq \ell$:

$$\boldsymbol{V}_\tau = \boldsymbol{\Gamma} \cdot e + \sum_{i=1}^{\tau-1} \beta_i \boldsymbol{V}_i$$

for some $\boldsymbol{\Gamma} = (\gamma_1, \ldots, \gamma_\ell) \in \mathbb{Z}^\ell$. That is,

$$v_{\tau,j} = \gamma_j \cdot e + \sum_{i=1}^{\tau-1} \beta_i \cdot v_{i,j}$$

Then using (1), one obtains :

$$\mu(m_\tau) = \prod_{j=1}^{\ell} p_j^{v_{\tau,j}} = \prod_{j=1}^{\ell} p_j^{\gamma_j \cdot e + \sum_{i=1}^{\tau-1} \beta_i \cdot v_{i,j}} = \left(\prod_{j=1}^{\ell} p_j^{\gamma_j}\right)^e \cdot \prod_{j=1}^{\ell}\prod_{i=1}^{\tau-1} p_j^{v_{i,j}\cdot\beta_i}$$

$$\mu(m_\tau) = \left(\prod_{j=1}^{\ell} p_j^{\gamma_j}\right)^e \cdot \prod_{i=1}^{\tau-1}\left(\prod_{j=1}^{\ell} p_j^{v_{i,j}}\right)^{\beta_i} = \left(\prod_{j=1}^{\ell} p_j^{\gamma_j}\right)^e \cdot \prod_{i=1}^{\tau-1} \mu(m_i)^{\beta_i}$$

That is:

$$\mu(m_\tau) = \delta^e \cdot \prod_{i=1}^{\tau-1} \mu(m_i)^{\beta_i}, \quad \text{where we denote: } \delta = \prod_{j=1}^{\ell} p_j^{\gamma_j} \tag{2}$$

Therefore, we see that $\mu(m_\tau)$ can be written as a multiplicative combination of the other $\mu(m_i)$. For RSA signatures, the attacker will ask for the signatures of $m_1, \ldots, m_{\tau-1}$ and forge the signature of m_τ using the relation:

$$\sigma_\tau = \mu(m_\tau)^d = \delta \cdot \prod_{i=1}^{\tau-1}\left(\mu(m_i)^d\right)^{\beta_i} = \delta \cdot \prod_{i=1}^{\tau-1} \sigma_i^{\beta_i} \mod N$$

In Appendix B we describe the corresponding forgery for Rabin-Williams signatures, where, in some cases, the attacker may even factor N.

The attack's complexity depends on ℓ and on the probability that the integers $\mu(m_i)$ are B-smooth. The reader is referred to Appendix C for a complexity analysis (see also [17]). In practice, the attack is feasible only if the $\mu(m_i)$ are relatively small (*e.g.* less than 200 bits).

4 Coron-Naccache-Stern's Attack

In ISO/IEC 9796-2, the encoding function's output $\mu(m)$ is as long as N. This thwarts Desmedt and Odlyzko's attack. Coron-Naccache-Stern's workaround [19] consisted in generating messages m_i such that a *linear combination* t_i of $\mu(m_i)$ and N is much smaller than N. Then, the attack can be applied to the integers t_i instead of $\mu(m_i)$.

More precisely, Coron *et al.* observed that it is sufficient to find a constant a and messages m_i such that:

$$t_i = a \cdot \mu(m_i) \mod N$$

is small, instead of requiring that $\mu(m_i)$ is small. Namely, the factor a can be easily dealt with by regarding $a^{-1} \bmod N$ as an "additional factor" in $\mu(m_i)$; to that end we only need to add one more column in the matrix considered in section 3. In their attack Coron *et al.* used $a = 2^8$.

Obtaining a small $a \cdot \mu(m) \bmod N$ is done in [19] as follows. From the definition of ISO/IEC 9796-2:

$$\begin{aligned}\mu(m) &= \mathtt{6A}_{16} \quad \| \; m[1] \quad \| \; \text{HASH}(m) \quad \| \; \mathtt{BC}_{16} \\ &= \mathtt{6A}_{16} \cdot 2^{k-8} + m[1] \cdot 2^{k_h+8} + \text{HASH}(m) \cdot 2^8 + \mathtt{BC}_{16}\end{aligned}$$

Euclidean division by N provides b and $0 \le r < N < 2^k$ such that:

$$(\mathtt{6A}_{16} + 1) \cdot 2^k = b \cdot N + r$$

Denoting $N' = b \cdot N$ one can write:

$$\begin{aligned} N' &= \texttt{6A}_{16} \cdot 2^k + \quad (2^k - r) \\ &= \texttt{6A}_{16} \qquad \| \; N'[1] \| N'[0] \end{aligned}$$

where N' is $k+7$ bits long and $N'[1]$ is $k - k_h - 16$ bits long.

Consider the linear combination:

$$\begin{aligned} t &= b \cdot N - a \;\; \cdot \mu(m) \\ &= N' \quad - 2^8 \cdot \mu(m) \end{aligned}$$

By setting $m[1] = N'[1]$ we get:

$$\begin{aligned} t = &\quad\; \texttt{6A}_{16} \,\|\, N'[1] \,\|\, N'[0] \\ &- \texttt{6A}_{16} \,\|\, m[1] \;\,\|\, \textsc{hash}(m) \| \texttt{BC00}_{16} \\ = &\quad\; \cancel{\texttt{6A}_{16}} \,\|\, \cancel{N'[1]} \,\|\, N'[0] \\ &- \cancel{\texttt{6A}_{16}} \,\|\, \cancel{N'[1]} \,\|\, \textsc{hash}(m) \| \texttt{BC00}_{16} \\ = &\qquad\qquad\quad N'[0] - (\textsc{hash}(m) \| \texttt{BC00}_{16}) < 2^{k_h + 16} \end{aligned}$$

For $k_h = 160$, the integer t is therefore at most 176-bits long.

The forger can thus modify $m[2]$ (and therefore $\textsc{hash}(m)$), until he gets a set of messages whose t-values are B-smooth and express one such $\mu(m_\tau)$ as a multiplicative combination of the others. As per the analysis in [19], attacking the instances $k_h = 128$ and $k_h = 160$ requires (respectively) 2^{54} and 2^{61} operations.

Note that the sign of the t_i's must be accounted for. This is simple because $(-1)^d \bmod N$ is public. Hence, when an odd number of t_i's is used a minus sign is inserted into δ.

5 The New Attack's Building-Blocks

We improve the above complexities by using four new ideas: we accelerate Desmedt-Odlyzko's process using Bernstein's smoothness detection algorithm [7], instead of trial division; we also use the large prime variant [1]; moreover, we modify Coron *et al.*'s attack by selecting better messages and by optimizing exhaustive search to equilibrate complexities. In this section we present these new building-blocks.

5.1 Bernstein's Smoothness Detection Algorithm

Bernstein [7] describes the following algorithm for finding smooth integers.

Algorithm: Given prime numbers $p_1, \ldots, p_\ell$ in increasing order and positive integers $t_1, \ldots, t_n$, output the p_ℓ-smooth part of each t_k:

1. Compute $z \leftarrow p_1 \times \cdots \times p_\ell$ using a product tree.
2. Compute $z_1 \leftarrow z \bmod t_1, \ldots, z_n \leftarrow z \bmod t_n$ using a remainder tree.
3. For each $k \in \{1, \ldots, n\}$: Compute $y_k \leftarrow (z_k)^{2^e} \bmod t_k$ by repeated squaring, where e is the smallest non-negative integer such that $2^{2^e} \geq t_k$.
4. For each $k \in \{1, \ldots, n\}$: output $\gcd(t_k, y_k)$.

We refer the reader to [6] for a description of the product and remainder trees.

Theorem 1 (Bernstein). *The algorithm computes the p_ℓ-smooth part of each t_k, in $\mathcal{O}(b \log^2 b \log\log b)$ time, where b is the number of input bits.*

In other words, given a list of n_t integers $t_i < 2^a$ and the list of the first ℓ primes, the algorithm will detect the B-smooth t_i's, where $B = p_\ell$, in complexity:

$$\mathcal{O}(b \cdot \log^2 b \cdot \log\log b)$$

where $b = n_t \cdot a + \ell \cdot \log_2 \ell$ is the total number of input bits.

When n_t is very large, it becomes more efficient to run the algorithm k times, on batches of $n'_t = n_t/k$ integers. We explain in Appendix D how to select the optimal n'_t, and derive the corresponding running time.

Bernstein recommends a number of speed-up ideas of which we used a few. In our experiments we used the *scaled remainder tree* [9], which replaces most division steps in the remainder tree by multiplications. This algorithm is fastest when FFT multiplications are done modulo numbers of the form $2^\alpha - 1$: we used this *Mersenne* FFT *multiplication* as well, as implemented in Gaudry, Kruppa and Zimmermann's GMP patch [24]. Other optimizations included computing the product z only once, and treating the prime 2 separately.

Bernstein's algorithm was actually the main source of the attack's improvement. It proved $\simeq 1000$ faster than the trial division used in [19].

5.2 The Large Prime Variant

An integer is *semi-smooth* with respect to y and z if its greatest prime factor is $\leq y$ and all other factors are $\leq z$. Bach and Peralta [1] define the function $\sigma(u, v)$, which plays for semi-smoothness the role played by Dickman's ρ function for smoothness (see Appendix C): $\sigma(u, v)$ is the asymptotic probability that an integer n is semi-smooth with respect to $n^{1/v}$ and $n^{1/u}$.

After an integer t_i has had all its factors smaller than B stripped-off, if the remaining factor ω is lesser than B^2 then ω must be prime. This is very easy to detect using Bernstein's algorithm. As Bernstein computes the B-smooth part z_i of each t_i, it only remains to check whether t_i/z_i is small enough. In most cases it isn't even necessary to perform the actual division since comparing the sizes of t_i and z_i suffices to rule out most non-semi-smooth numbers.

Hence, one can use a second bound B_2 such that $B < B_2 < B^2$ and keep the t_i's whose remaining factor ω is $\leq B_2$, hoping to find a second t_i with the same remaining factor ω to divide ω out. We refer the reader to Appendix E for a detailed analysis of the large prime variant in our context.

5.3 Constructing Smaller $a \cdot \mu(m) - b \cdot N$ Candidates

In this paragraph we show how to construct smaller $t_i = a \cdot \mu(m_i) - b \cdot N$ values for ISO/IEC 9796-2. Smaller t_i-values increase smoothness probability and hence accelerate the forgery process.

We write:

$$\mu(x, h) = \texttt{6A}_{16} \cdot 2^{k-8} + x \cdot 2^{k_h+8} + h \cdot 2^8 + \texttt{BC}_{16}$$

where $x = m[1]$ and $h = \text{HASH}(m)$, with $0 < x < 2^{k-k_h-16}$.

We first determine $a, b > 0$ such that the following two conditions hold:

$$0 < b \cdot N - a \cdot \mu(0, 0) < a \cdot 2^{k-8} \quad (3)$$

$$b \cdot N - a \cdot \mu(0, 0) = 0 \mod 2^8 \quad (4)$$

and a is of minimal size. Then by Euclidean division we compute x and r such that:

$$b \cdot N - a \cdot \mu(0, 0) = (a \cdot 2^{k_h+8}) \cdot x + r$$

where $0 \leq r < a \cdot 2^{k_h+8}$ and using (3) we have $0 \leq x < 2^{k-k_h-16}$ as required. This gives:

$$b \cdot N - a \cdot \mu(x, 0) = b \cdot N - a \cdot \mu(0, 0) - a \cdot x \cdot 2^{k_h+8} = r$$

Moreover as per (4) we must have $r = 0 \mod 2^8$; denoting $r' = r/2^8$ we obtain:

$$b \cdot N - a \cdot \mu(x, h) = r - a \cdot h \cdot 2^8 = 2^8 \cdot (r' - a \cdot h)$$

where $0 \leq r' < a \cdot 2^{k_h}$. We then look for smooth values of $r' - a \cdot h$, whose size is at most k_h plus the size of a.

If a and b are both 8-bit integers, this gives 16 bits of freedom to satisfy both conditions (3) and (4); heuristically each of the two conditions is satisfied with probability $\simeq 2^{-8}$; therefore, we can expect to find such an $\{a, b\}$ pair. For example, for the RSA-2048 challenge, we found $\{a, b\}$ to be $\{625, 332\}$; therefore, for RSA-2048 and $k_h = 160$, the integer to be smooth is 170-bits long (instead of 176-bits in Coron *et al.*'s original attack). This decreases further the attack's complexity. In Appendix F we provide the optimal $\{a, b\}$ values for other RSA challenge moduli.

6 Attacking ISO/IEC 9796-2

We combined all the building-blocks listed in the previous section to compute an actual forgery for ISO/IEC 9796-2, with the RSA-2048 challenge modulus. The implementation replaced Coron *et al.*'s trial division by Bernstein's algorithm, replaced Coron *et al.*'s $a \cdot \mu(m) - b \cdot N$ values by the shorter t_i's introduced in paragraph 5.3 and took advantage of the large prime variant. Additional speed-up was obtained by exhaustive searching for particular digest values. Code was written in C++ and run on 19 Linux-based machines on the Amazon EC2 grid. The final linear algebra step was performed on a single PC.

6.1 The Amazon Grid

Amazon.com Inc. offers virtualized computer instances for rent on a pay by the hour basis, which we found convenient to run our computations. Various models are available, of which the best-suited for CPU-intensive tasks, as we write these lines, features 8 Intel Xeon 64-bit cores clocked at 2.4GHZ supporting the Core2 instruction set and offering 7GB RAM and 1.5TB disk space. Renting such a capacity costs US\$0.80 per hour (plus tax). One can run up to 20 such instances in parallel, and possibly more subject to approval by Amazon (20 were enough for our purpose so we didn't apply for more).

When an instance on the grid is launched, it starts up from a disk image containing a customizable UNIX operating system. In the experiment, we ran a first instance using the basic Fedora installation provided by default, installed necessary tools and libraries, compiled our own programs and made a disk image containing our code, to launch subsequent instances with. When an instance terminates, its disk space is freed, making it necessary to save results to some permanent storage means. We simply `rsync`'ed results to a machine of ours. Note that Amazon also charges for network bandwidth but data transmission costs were negligible in our case.

All in all, we used about 1,100 instance running hours (including setup and tweaks) during a little more than two days. While we found the service to be rather reliable, one instance failed halfway through the computation, and its intermediate results were lost.

6.2 The Experiment: Outline, Details and Results

The attack can be broken down into the following elementary steps, which we shall review in turn:

1. Determining the constants $a, b, x, \mu(x, 0)$ for the RSA-2048 challenge modulus N.
2. Computing the product of the first ℓ primes, for a suitable choice of ℓ.
3. Computing the integers $t_i = bN - a\mu(m_i)$, and hence the SHA-1 digests, for sufficiently many messages m_i.
4. Finding the smooth and semi-smooth integers amongst the t_i's.
5. Factoring the smooth integers, as well as the colliding pairs of semi-smooth integers, obtaining the sparse, singular matrix of exponents, with ℓ rows and more than ℓ columns.
6. Reducing this matrix modulo $e = 2$, with possible changes in the first row (corresponding to the prime 2) depending on the Jacobi symbols $(2|t_i)$ and $(2|a)$.
7. Finding nontrivial vectors in the kernel of this reduced matrix and inferring a forgery.

Steps 2–4 were executed on the Amazon EC2 grid, whereas all other steps were run on one offline PC. Steps 3–4, and to a much lesser extent step 7, were the only steps that claimed a significant amount of CPU time.

Determining the constants. The attack's complexity doesn't depend on the choice of N. Since N has to be congruent to $5 \bmod 8$ for Rabin-Williams signatures, we used the RSA-2048 challenge. The resulting constants were computed in SAGE [50]. We found the smallest $\{a, b\}$ pair to be $\{625, 332\}$, and the $\mu(x, 0)$ value given in Appendix G. The integers $t_i = bN - a\mu(x, h_i)$ are thus at most 170-bits long.

Product of the first primes. The optimal choice of ℓ for 170 bits is about 2^{21}. Since the Amazon instances are memory-constrained (less than 1GB of RAM per core), we preferred to use $\ell = 2^{20}$. This choice had the additional advantage of making the final linear algebra step faster, which is convenient since this step was run on a single off-line PC. Computing the product of primes itself was done once and for all in a matter of seconds using MPIR.

Hashing. Since the attack's smoothness detection part works on batches of t_i's (in our cases, we chose batches of 2^{19} integers), we had to compute digests of messages m_i in batches as well. The messages themselves are 2048-bit long, *i.e.* as long as N, and comply with the structure indicated in Appendix J: a constant 246-byte prefix followed by a 10-byte seed. The first two bytes identify a family of messages examined on a single core of one Amazon instance, and the remaining eight bytes are explored by increments of 1 starting from 0.

Messages were hashed using OpenSSL's SHA-1 implementation. For each message, we only need to compute one SHA-1 block, since the first three 64-byte blocks are fixed. This computation is relatively fast compared to Bernstein's algorithm, so we have a bit of leeway for exhaustive search. We can compute a large number of digests, keeping the ones likely to give rise to a smooth t_i. We did this by selecting digests for which the resulting t_i would have many zeroes as leading and trailing bits.

More precisely, we looked for a particular bit pattern at the beginning and at the end of each digest h_i, such that finding n matching bits results in n null bits at the beginning and at the end of t_i. The probability of finding n matching bits when we add the number of matches at the beginning and at the end is $(1 + n/2) \cdot 2^{-n}$, so we expect to compute $2^n/(1 + n/2)$ digests per selected message. We found $n = 8$ to be optimal: on average, we need *circa* 50 digests to find a match, and the resulting t_i is at most $170 - 8 = 162$ bit long (once powers of 2 are factored out).

Note that faster (*e.g.* hardware-enhanced) ways to obtain digests can significantly reduce the attack's complexity (*cf.* Appendix M). We considered for example an FPGA-based solution called COPACOBANA [46], which could in principle perform a larger amount of exhaustive search, and accelerate the attack dramatically. It turned out that our attack was fast enough, hence pursuing the hardware-assisted search idea further proved unnecessary, but a practical attack on EMV (*cf.* section 8) could certainly benefit from hardware acceleration.

Finding smooth and semi-smooth integers Once a batch of 2^{19} appropriate t_i's is generated, we factor out powers of 2, and feed the resulting odd numbers into our C++ implementation of Bernstein's algorithm. This implementation uses the MPIR multi-precision arithmetic library [26], which we chose over vanilla GMP because of a number of speed improvements, including J.W. Martin's patch for the Core2 architecture. We further applied Gaudry, Kruppa and Zimmermann's FFT patch,[2] mainly for their implementation of *Mersenne* FFT multiplication, which is useful in the scaled remainder tree [9].

We looked for B-smooth as well as for (B, B_2)-semi-smooth t_i's, where $B = 16{,}290{,}047$ is the 2^{20}-th prime, and $B_2 = 2^{27}$. Each batch took $\simeq 40$ seconds to generate and to process, and consumed about 500MB of memory. We ran 8 such processes in parallel on each instance to take advantage of the 8 cores, and 19 instances simultaneously.

The entire experiment can be summarized as follows:

$$\begin{tikzcd}
& \substack{\text{16,230,259,553,940}\\\text{digest computations}} \arrow[d] & \\
& \substack{\text{339,686,719,488 } t_i\text{'s in}\\\text{647,901 batches of } 2^{19} \text{ each}} \arrow[dl] \arrow[dr] & \\
\substack{\text{684,365}\\\text{smooth } t_i\text{'s}} \arrow[dr] & & \substack{\text{366,302 collisions between}\\\text{2,786,327 semi-smooth } t_i\text{'s}} \arrow[dl] \\
& \text{1,050,667-column matrix} \arrow[d] & \\
& \substack{\text{algebra on 839,908 columns}\\\text{having} > 1 \text{ nonzero entry}} \arrow[d] & \\
& \text{124 kernel vectors} \arrow[d] & \\
& \text{forgery involving 432,903 signatures} &
\end{tikzcd}$$

Finding the 1,050,667 columns (slightly in excess of the $\ell = 2^{20} = 1{,}048{,}576$ required) took a little over 2 days.

Factoring and finding collisions. The output of the previous stage is a large set of text files containing the smooth and semi-smooth t_i's together with the corresponding message numbers. Turning this data

[2] A colleague suggested [41] that our implementation could be further improved by using the floating-point DWT multiplication algorithm described in [20] and implemented in GIMPS, which is reportedly up to 5 or 10 times faster than integer DWT depending on the architecture.

into a matrix suitable for the linear algebra stage mostly involved text manipulation in Perl to convert it to commands that could be piped into PARI/GP [47]. The resulting 1,048,576 × 1,050,667 matrix had 14,215,602 non-zero entries (13.5 per column on average, or 10^{-5} sparsity; the columns derived from the large prime variant tend to have twice as many non-zero entries, of course).[3]

Linear algebra. We found non-zero kernel elements of the final sparse matrix over GF(2) using Coppersmith's block Wiedemann algorithm [13] implemented in WLSS2 [33, 39], with parameters $m = n = 4$ and $\kappa = 2$. The whole computation took 16 hours on one 2.7GHz personal computer, with the first (and longest) part of the computation using 2 cores, and the final part using 4 cores.

The program discovered 124 kernel vectors with Hamming weights ranging from 337,458 to 339,641. Since columns obtained from pairs of semi-smooth numbers account for two signatures each, the number of signature queries required to produce the 124 corresponding forgeries is slightly larger, and ranges between 432,903 and 435,859.

Being written with the quadratic sieve in mind, the block Wiedemann algorithm in WLSS2 works over GF(2). There exist, however, other implementations for different finite fields.

Evidencing forgery. An interesting question is that of exhibiting a *compact evidence of forgery*. In Appendix J we exhibit statistical evidence that a multiplicative relation between ISO/IEC 9796-2 signatures, (*i.e.* a forgery) was indeed constructed.

Fewer signature queries In Appendix K we address the question of reducing the number of signature queries in the attack.

7 Cost Estimates

The experiment described in the previous section can be used as a benchmark to estimate the attack's cost as a function of the size of the t_i's, denoted a; this will be useful for analyzing the security of the EMV specifications, where a is bigger (204 bits instead of 170 bits).

$a = \log_2 t_i$	$\log_2 \ell$	Estimated TotalTime	$\log_2 \tau$	Amazon EC2 cost (US$)
64	11	15 seconds	20	negligible
128	19	4 days	33	10
160	21	6 months	38	470
170	22	1.8 years	40	1,620
176	23	3.8 years	41	3,300
204	25	95 years	45	84,000
232	27	19 centuries	49	1,700,000
256	30	320 centuries	52	20,000,000

Table 1. Bernstein+Large prime variant. Estimated parameter trade-offs, running times and costs, for various t_i sizes.

[3] This matrix actually contained a number of rows with only one nonzero entry or less. Those rows and the corresponding columns can be safely removed, and the process can be repeated on the resulting matrix until a fix-point is reached (*cf.* Appendix L for details). This reduction process is frequently the first operation carried out by linear algebra packages when searching for kernel vectors. When applied to our matrix, it produced a reduced matrix of dimension 750,031 × 839,908. We found it most convenient to leave any such reduction step to the linear algebra package itself.

Results are summarized in Table 1. We assume that the t_i's are uniformly distributed a-bit integers and express costs as a function of a. Cost figures do not include the linear algebra step whose computational requirements are very low compared to the smoothness detection step. Another difference with our experiment is that here we do not assume any exhaustive search on the t_i's; this is why the cost estimate for $a = 170$ in Table 1 is about the double of the cost of our experimental ISO/IEC 9796-2 forgery.

Running times are given for a single 2.4GHz PC. Costs correspond to the Amazon EC2 grid, as in the previous section. Estimates show that the attack is feasible up to $\simeq 200$ bits, but becomes infeasible for larger values of a. We also estimate $\log_2 \tau$, where τ is the number of messages in the forgery.

8 Application to EMV Signatures

EMV is a collection of industry specifications for the inter-operation of payment cards, POS terminals and ATMs. The name EMV is the acronym of the initial letters of Europay, MasterCard and Visa, the three companies which originally cooperated to develop the specifications. Europay International SA was absorbed into Mastercard in 2002. JCB (formerly Japan Credit Bureau) joined the organization in December 2004, and American Express joined in February 2009.

The EMV specifications [23] rely on ISO/IEC 9796-2 signatures to certify public-keys and to authenticate data. For instance, when an Issuer provides application data to a Card, this data must be signed using the Issuer's private key S_I. The corresponding public-key P_I must be signed by a Certification Authority (CA) whose public-key is denoted P_CA. The signature algorithm is RSA with $e = 3$ or $e = 2^{16} + 1$. The bit length of all moduli is always a multiple of 8.

EMV uses special message formats; 7 different formats are used, depending on the message type. We first describe one of these formats: the *Static Data Authentication, Issuer Public-key Data* (SDA-IPKD), and adapt our attack to it. We then consider the other six formats.

8.1 EMV Static Data Authentication, Issuer Public-key Data (SDA-IPKD)

We refer the reader to §5.1, Table 2, page 41 in EMV [23]. SDA-IPKD is used by the CA to sign the issuer's public-key P_I. The message to be signed is as follows:

$$m = \texttt{02}_{16} \| X \| Y \| N_\text{I} \| \texttt{03}_{16}$$

where X represents 6 bytes that can be controlled by the adversary and Y represents 7 bytes that cannot be controlled. N_I is the Issuer's modulus to be certified. More precisely, $X = \text{ID} \| \text{DATE}$ where ID is the issuer identifier (4 bytes) and DATE is the *Certificate Expiration Date* (2 bytes); we assume that both can be controlled by the adversary. $Y = \text{CSN} \| C$ where CSN is the 3-bytes *Certificate Serial Number* assigned by the CA and C is a constant. Finally, the modulus to be certified N_I can also be controlled by the adversary.

With ISO/IEC 9796-2 encoding, this gives:

$$\mu(m) = \texttt{6A02}_{16} \| X \| Y \| N_{\text{I},1} \| \text{HASH}(m) \| \texttt{BC}_{16}$$

where $N_\text{I} = N_{\text{I},1} \| N_{\text{I},2}$ and the size of $N_{\text{I},1}$ is $k - k_h - 128$ bits. k denotes the modulus size and $k_h = 160$ as in ISO/IEC 9796-2.

8.2 Attacking SDA-IPKD

To attack SDA-IPKD write:

$$\mu(X, N_{\mathrm{I},1}, h) = \mathtt{6A02}_{16} \cdot 2^{k_1} + X \cdot 2^{k_2} + Y \cdot 2^{k_3} + N_{\mathrm{I},1} \cdot 2^{k_4} + h$$

where Y is constant and $h = \text{HASH}(m) \| \mathtt{BC}_{16}$. We have:

$$\begin{cases} k_1 = k \quad - 16 \\ k_2 = k_1 - 48 = k - 64 \\ k_3 = k_2 - 56 = k - 120 \\ k_4 = k_h + \ 8 \ = 168 \end{cases}$$

Generate a random k_a-bit integer a, where $36 \le k_a \le 72$, and consider the equation:

$$b \cdot N - a \cdot \mu(X, 0, 0) = b \cdot N - a \cdot X \cdot 2^{k_2} - a \cdot (\mathtt{6A02}_{16} \cdot 2^{k_1} + Y \cdot 2^{k_3})$$

If we can find integers X and b such that $0 \le X < 2^{48}$ and:

$$0 \le b \cdot N - a \cdot \mu(X, 0, 0) < a \cdot 2^{k_3} \tag{5}$$

then as previously we can compute $N_{\mathrm{I},1}$ by Euclidean division:

$$b \cdot N - a \cdot \mu(X, 0, 0) = (a \cdot 2^{k_4}) \cdot N_{\mathrm{I},1} + r \tag{6}$$

where $0 \le N_{\mathrm{I},1} < 2^{k_3 - k_4}$ as required, and the resulting $b \cdot N - a \cdot \mu(X, N_{\mathrm{I},1}, h)$ value will be small for all values of h.

In the above we assumed Y to be a constant. Actually the first 3 bytes of Y encode the CSN assigned by the CA, and may be different for each new certificate (see Appendix H). However if the attacker can predict the CSN, then he can compute a different a for every Y and adapt the attack by factoring a into a product of small primes.

Finding small X and b so as to minimize the value of

$$|b \cdot N - a \cdot X \cdot 2^{k_2} - a \cdot (\mathtt{6A02}_{16} \cdot 2^{k_1} + Y \cdot 2^{k_3})|$$

is a Closest Vector Problem (CVP) in a bi-dimensional lattice; a problem that can be easily solved using the LLL algorithm [35]. We first determine heuristically the minimal size that can be expected; we describe the LLL attack in Appendix H.

Since the $a \cdot \mathtt{6A02}_{16} \cdot 2^{k_1}$ is an $(k + k_a)$-bit integer, with $X \simeq 2^{48}$ and $b \simeq 2^{k_a}$, odds are heuristically reasonable to find X and b such that:

$$0 \le b \cdot N - a \cdot \mu(X, 0, 0) < 2^{(k+k_a)-48-k_a} = 2^{k-48} \simeq a \cdot 2^{k-48-k_a} = a \cdot 2^{k_3+72-k_a}$$

which is $(72 - k_a)$-bit too long compared to condition (5). Therefore, by exhaustive search we will need to examine roughly 2^{72-k_a} different integers a to find a pair (b, X) that satisfies (5); since a is k_a-bits long, this can be done only if $72 - k_a \le k_a$, which gives $k_a \ge 36$. For $k_a = 36$ we have to exhaust the 2^{36} possible values of a.

Once this is done we obtain from (6):

$$t = b \cdot N - a \cdot \mu(X, N_{\mathrm{I},1}, h) = r - a \cdot h$$

with $0 \leq r < a \cdot 2^{k_4}$. This implies that the final size of t values is $168 + k_a$ bits. For $k_a = 36$ this gives 204 bits (instead of 170 bits for pure ISO/IEC 9796-2). The attack's complexity will hence be higher than for plain ISO/IEC 9796-2.

In Appendix H we exhibit concrete (a, b, X) values for $k_a = 52$ and for the RSA-2048 challenge; this required $\simeq 2^{23}$ trials (109 minutes on a single PC). We estimate that for $k_a = 36$ this computation will take roughly 13 years on a single PC, or equivalently US$11,000 using the EC2 grid.

Table 1 shows that attacking 204-bit t_i's would cost $\simeq$ US$84,000. As for the ISO/IEC 9796-2 attack, we can decrease this cost by first doing exhaustive search on the bits of HASH(m) to obtain a smaller t-value. We found that with 8 bits of exhaustive search cost drops to $\simeq$ US$45,000 (without the matrix step, but in our attack algebra takes a relatively small amount of time).

8.3 Summary

In Appendix I we provide an analysis of the other formats in the EMV specifications, with corresponding attacks when such attacks exist. We summarize results in Table 2 where an X represents a string that can be controlled by the adversary, while Y cannot be controlled. The size of the final t-value to be smooth is given in bits. Note that cost estimates are cheaper than Table 1 because we first perform exhaustive search on 8 bits of HASH(m) = SHA-1(m); however here we do take into account the cost of computing these SHA-1(m) values.

EMV mode	Format	Size of X	Size of Y	Size of t	EC2 cost (US$)
SDA-IPKD	$02_{16} \| X \| Y \| N_I \| 03_{16}$	48	56	204	45,000
SDA-SAD	Y	-	$k - 176$	-	-
ODDA-IPKD	$02_{16} \| X \| Y \| N_I \| 03_{16}$	48	56	204	45,000
ODDA-ICC-PKD	$04_{16} \| X \| Y \| N_{ICC} \| 03_{16} \| \mathtt{data}$	96	56	204	45,000
ODDA-DAD1	Y	-	$k - 176$	-	-
ODDA-DAD2	Y	-	$k - 176$	-	-
ICC-PIN	$04_{16} \| X \| Y \| N_{ICC} \| 03_{16}$	96	56	204	45,000

Table 2. Various EMV message formats. X denotes a data field controllable by the adversary. Y is not controllable. Data sizes for X, Y and t are expressed in bits.

Table 2 shows that only four of the EMV formats can be attacked, with the same complexity as the SDA-IPKD format. The other formats seem out of reach because the non-controllable part Y is too large.

Note that these attacks do not threaten any of the 730 million EMV payment cards in use worldwide for *operational* reasons: the Issuer and the CA will never accept to sign the chosen messages necessary for conducting the attack.

9 Conclusion

This paper exhibited a *practically exploitable* flaw in the *currently valid* ISO/IEC 9796-2 standard and a conceptual flaw in EMV signatures. The authors recommend the definite withdrawal of the *ad-hoc* encoding mode in ISO/IEC 9796-2 and its replacement by a provably secure encoding function such as PSS.

References

1. E. Bach and R. Peralta, *Asymptotic semismoothness probabilities*, Mathematics of Computation, vol. 65, number 216, 1996, pp. 1701–1715.
2. M. Bellare and P. Rogaway, *Random oracles are practical: A paradigm for designing efficient protocols*, Proceedings of CCS 1993, ACM, 1993, pp. 62–73.
3. M. Bellare and P. Rogaway, *Optimal Asymmetric Encryption: How to encrypt with* RSA, Proceedings of Eurocrypt 1994, LNCS, vol. 950, Springer-Verlag, 1995, pp. 92–111.
4. M. Bellare and P. Rogaway, *The Exact security of digital signatures: How to sign with* RSA *and Rabin*, Proceedings of Eurocrypt 1996, LNCS, vol. 1070, Springer-Verlag, 1996, pp. 399–416.
5. D.J. Bernstein and T. Lange (editors), *eBACS:* ECRYPT *Benchmarking of cryptographic systems*, `bench.cr.yp.to`.
6. D.J. Bernstein, *Fast Multiplications and its applications*, Algorithmic Number Theory, vol. 44, 2008.
7. D.J. Bernstein, *How to find smooth parts of integers*, 2004/05/10, `cr.yp.to/papers.html\#smoothparts`.
8. D.J. Bernstein, *Proving tight security for Rabin-Williams signatures*. Proceedings of Eurocrypt 2008, LNCS, vol. 4665, Springer-Verlag, 2008, pp. 70–87.
9. D.J. Bernstein, *Scaled remainder trees*, 2004/08/20, `cr.yp.to/papers.html\#scaledmod`.
10. D.J. Bernstein, T. Lange and Ch. Peters, *Attacking and defending the McEliece cryptosystem*, Proceedings of Post-Quantum Cryptography 2008, LNCS, vol. 5299, Springer-Verlag, 2008, pp. 31–46.
11. D. Bleichenbacher, *Chosen ciphertext attacks against protocols based on the* RSA *encryption standard*, Proceedings of Crypto 1998, LNCS, vol. 1462, Springer-Verlag, 1998, pp. 1–12.
12. E.R. Canfield, P. Erdős and C. Pomerance, *On a Problem of Oppenheim concerning 'Factorisation Numerorum'*, Journal of Number Theory, vol. 17, 1983, pp. 1–28.
13. D. Coppersmith, *Solving homogeneous linear equations over* GF(2) *via block Wiedemann algorithm*, Mathematics of Computation, vol. 62, number 205, 1994, pp. 333–350.
14. D. Coppersmith, J.-S. Coron, F. Grieu, S. Halevi, C.S. Jutla, D. Naccache and J.P. Stern, *Cryptanalysis of* ISO/IEC *9796-1*, Journal of Cryptology, vol. 21, 2008, pp. 27–51.
15. D. Coppersmith, S. Halevi and C. Jutla, ISO 9796-1 *and the new, forgery strategy*, Research contribution to P.1363, 1999, `grouper.ieee.org/groups/1363/Research`.
16. J.-S. Coron, *Security proofs for partial domain hash signature schemes*, Proceedings of Crypto 2002, LNCS, vol. 2442, Springer-Verlag, 2002, pp. 613–626.
17. J.-S. Coron, Y. Desmedt, D. Naccache, A. Odlyzko and J.P. Stern, *Index calculation attacks on* RSA *signature and encryption* Designs, Codes and Cryptography, vol. 38, number 1, 2006, pp. 41–53.
18. J.-S. Coron, D. Naccache, M. Joye and P. Paillier, *New attacks on* PKCS*#1 v1.5 encryption*, Proceedings of Eurocrypt 2000, LNCS, vol. 1807, Springer-Verlag, 2000, pp. 369–381.
19. J.-S. Coron, D. Naccache and J.P. Stern, *On the security of* RSA *padding*, Proceedings of Crypto 1999, LNCS, vol. 1666, Springer-Verlag, 1999, pp. 1–18.
20. R.E. Crandall, E.W. Mayer and J.S. Papadopoulos, *The twenty-fourth Fermat number is composite*, Mathematics of Computation, volume 72, number 243, July 2003, pp. 1555–1572.
21. Y. Desmedt and A. Odlyzko. *A chosen text attack on the* RSA *cryptosystem and some discrete logarithm schemes*, Proceedings of Crypto 1985, LNCS, vol. 218, Springer-Verlag, 1986, pp. 516–522.
22. K. Dickman, *On the frequency of numbers containing prime factors of a certain relative magnitude*, Arkiv för matematik, astronomi och fysik, vol. 22A, no. 10, 1930, pp. 1–14.
23. EMV, *Integrated circuit card specifications for payment systems*, Book 2. Security and Key Management. Version 4.2. June 2008. `www.emvco.com`.
24. P. Gaudry, A. Kruppa and P. Zimmermann, *A gmp-based implementation of Schőnhage-Strassen's large integer multiplication algorithm*, in Proceedings of ISSAC 2007, Waterloo, Ontario, Canada, ACM Press, 2007, pp. 167–174.
25. F. Grieu, *A chosen messages attack on the* ISO/IEC *9796-1 signature scheme*, Proceedings of Eurocrypt 2000, LNCS, vol. 1807, Springer-Verlag, 2000, pp. 70–80.
26. W.B. Hart *et al.*, *Multiple Precision Integers and Rationals*, `www.mpir.org`.
27. W.B. Hart, D. Harvey *et al.*, *Fast Library for Number Theory*, `www.flintlib.org`.
28. ISO/IEC 9796, *Information technology – Security techniques – Digital signature scheme giving message recovery, Part 1: Mechanisms using redundancy*, 1999.
29. ISO/IEC 9796-2, *Information technology – Security techniques – Digital signature scheme giving message recovery, Part 2: Mechanisms using a hash-function*, 1997.
30. ISO/IEC 9796-2:2002, *Information technology – Security techniques – Digital signature schemes giving message recovery – Part 2: Integer factorization based mechanisms*, 2002.

31. A. Joux, D. Naccache, E. Thomé, *When e-th roots become easier than factoring*, Proceedings of Asiacrypt 2007, LNCS, vol. 4833, Springer-Verlag, 2007, pp. 13–28.
32. B. Kaliski, PKCS*#1:* RSA *Encryption Standard, Version 1.5*, RSA Laboratories, November 1993.
33. E. Kaltofen and A. Lobo, *Distributed matrix-free solution of large sparse linear systems over finite fields*, Algorithmica, vol. 24, 1999, pp. 331–348.
34. C. Lanczos, *An iterative method for the solution of the eigenvalue problem of linear differential and integral operator*, Journal of Research of the National Bureau of Standards, vol. 45, 1950, pp. 255–282.
35. A.K. Lenstra, H.W. Lenstra, Jr., and L. Lovász, *Factoring polynomials with rational coefficients.* Mathematische Annalen, vol. 261, 1982, pp. 513–534.
36. A.K. Lenstra and H.W. Lenstra, Jr., *The Development of the number field sieve*, Berlin: Springer-Verlag, 1993.
37. H. Lenstra, Jr., *Factoring integers with elliptic curves*, Annals of Mathematics, vol. 126, number 2, 1987, pp. 649–673.
38. Y. Liu, T. Kasper, K. Lemke-Rust, C. Paar: *E-Passport: Cracking basic access control keys*, OTM Conferences (2), 2007, pp. 1531-1547.
39. A. Lobo, WLSS*2: an implementation of the homogeneous block Wiedemann algorithm*, `www4.ncsu.edu/\~kaltofen/software/wiliss`.
40. A.J. Menezes, P.C. van Oorschot and S.A. Vanstone, *Handbook of applied cryptography*, CRC press, 1996.
41. M. Mezzarobba, *de auditu*, March 2009.
42. J.-F. Misarsky, *How (not) to design* RSA *signature schemes*, Proceedings of Public Key Cryptography 1998, LNCS, vol. 1431, Springer-Verlag, 1998, pp. 14–28.
43. P.L. Montgomery, *A block Lanczos algorithm for finding dependencies over* GF(2), Proceedings of Eurocrypt 1995, LNCS, vol. 921, Springer-Verlag, 1995, pp. 106–120.
44. NVIDIA, CUDA *Zone – The resource for* CUDA *developers*, `www.nvidia.com/cuda`.
45. D.A. Osvik, *de auditu*, March 2009.
46. C. Paar and M. Schimmer, COPACOBANA: *A Codebreaker for* DES *and other ciphers*, `www.copacobana.org`.
47. The PARI Group, *PARI/GP*, version `2.3.4`, Bordeaux, 2008, `pari.math.u-bordeaux.fr`.
48. C. Pomerance, *The quadratic sieve factoring algorithm*, Proceedings of Eurocrypt 1984, LNCS, vol. 209, Springer-Verlag, 1985, pp. 169–182.
49. R. Rivest, A. Shamir and L. Adleman, *A method for obtaining digital signatures and public key cryptosystems*, Communications of the ACM, vol. 21, 1978, pp. 120–126.
50. W.A. Stein et al., *Sage Mathematics Software (Version 3.3)*, The Sage Development Team, 2009, `www.sagemath.org`.
51. D. Stinson, *Cryptography: theory and practice*, 3-rd edition, CRC Press, 2005.
52. M. Stevens, A. Sotirov, J. Appelbaum, A. Lenstra, D. Molnar, D.A. Osvik, B. de Weger: *Short chosen-prefix collisions for* MD5 *and the creation of a rogue* CA *certificate*, Cryptology ePrint Archive, Report 2009/111, 2009.
53. V. Shoup, *Number Theory C++ Library (*NTL*) version 5.3.1.* `www.shoup.net/ntl`.

A Rabin-Williams Signatures

Rabin-Williams signatures use an encoding function $\mu(m)$ such that for all m, $\mu(m) = 12 \mod 16$.

– In contrast with RSA, it is required that $p = 3 \mod 8$ and $q = 7 \mod 8$.
  Since $e = 2$, the private key is $d = (N - p - q + 5)/8$.
– To sign a message m, compute the Jacobi symbol

$$\mathrm{J} = \left(\frac{\mu(m)}{N}\right)$$

The signature of m is $s = \min(\sigma, N - \sigma)$, where:

$$\sigma = \begin{cases} \mu(m)^d \mod N & \text{if } \mathrm{J} = 1 \\ (\mu(m)/2)^d \mod N & \text{otherwise} \end{cases}$$

– To verify the signature σ compute $\omega = s^2 \mod N$ and check that:

$$\mu(m) \overset{?}{=} \begin{cases} \omega & \text{if } \omega = 4 \mod 8 \\ 2 \cdot \omega & \text{if } \omega = 6 \mod 8 \\ N - \omega & \text{if } \omega = 1 \mod 8 \\ 2 \cdot (N - \omega) & \text{if } \omega = 7 \mod 8 \end{cases}$$

We recall here some known facts about Legendre and Jacobi symbols.

The Legendre symbol with respect to an odd prime p is defined by:

$$\left(\frac{x}{p}\right) = \begin{cases} 1 & \text{if } x \neq 0 \mod p \text{ and } x \text{ is a square modulo } p \\ 0 & \text{if } x = 0 \mod p \\ -1 & \text{otherwise.} \end{cases}$$

Lemma 1. *Let $p \neq 2$ be a prime. For any integer x,*

$$\left(\frac{x}{p}\right) = x^{\frac{p-1}{2}} \mod p$$

The Jacobi symbol with respect to an odd integer $n = \prod p_i^{e_i}$ is defined from Legendre symbols as follows:

$$\left(\frac{x}{n}\right) = \prod_i \left(\frac{x}{p_i}\right)^{e_i}$$

The Jacobi symbol can be computed without knowing the factorization of n; we refer the reader to [51] for more details.

The following lemma shows that the Rabin-Williams signature verification works. In particular, the fact that $\left(\frac{2}{N}\right) = -1$ ensures that either $\mu(m)$ or $\mu(m)/2$ has a Jacobi symbol equal to 1.

Lemma 2. *Let N be an* RSA*-modulus with $p = 3 \mod 8$ and $q = 7 \mod 8$. Then $\left(\frac{2}{N}\right) = -1$ and $\left(\frac{-1}{N}\right) = 1$. Let $d = (N - p - q + 5)/8$. Then for any integer x such that $\left(\frac{x}{N}\right) = 1$, we have that $x^{2d} = x \mod N$ if x is a square modulo N, and $x^{2d} = -x \mod N$ otherwise.*

We refer the reader to [40, 8] for more details on Rabin-Williams signatures.

B Desmedt and Odlyzko's Attack for Rabin-Williams Signatures

Let $\mathrm{J}(x)$ denote the Jacobi symbol of x with respect to N. For Rabin-Williams ($e = 2$), we distinguish two sub-cases:

$\mathbf{J}(\boldsymbol{\delta}) = \mathbf{1}$: we have $\delta^{2d} = \pm\delta \mod N$, which gives from (2) the forgery equation:

$$\mu(m_\tau)^d = \pm\delta \cdot \prod_{i=1}^{\tau-1} \left(\mu(m_i)^d\right)^{\beta_i} \mod N$$

$\mathbf{J}(\boldsymbol{\delta}) = \mathbf{-1}$: letting $u = \delta^{2d} \mod N$ we have:

$$u^2 = \delta^2 \mod N \Rightarrow (u-\delta)(u+\delta) = 0 \mod N$$

Since $\mathrm{J}(\delta) = -\mathrm{J}(u)$ we have $\delta \neq \pm u \mod n$ and $\gcd(u \pm \delta, N)$ will factor N. The attacker can therefore submit the τ messages for signature, recover $u = \delta^{2d}$, factor N and subsequently sign any message.

Note that in both cases we have assumed that the signature is always $\sigma = \mu(m)^d \mod N$, whereas by definition a Rabin-Williams signature is $\sigma = (\mu(m)/2)^d \mod N$ when $\mathrm{J}(\mu(m)) = -1$. A possible work-around consists in discarding messages for which $\mathrm{J}(\mu(m)) = -1$ but it is also easy to adapt the attack to handle both cases.

C Desmedt-Odlyzko's Attack: Complexity Analysis

The attack's complexity depends on ℓ and on the probability that the integers $\mu(m_i)$ are y-smooth. We define $\psi(x,y) = \#\{v \leq x$, such that v is y-smooth$\}$. It is known [22] that, for large x, the ratio $\psi(x, \sqrt[t]{x})/x$ is equivalent to Dickman's function defined by :

$$\rho(t) = \begin{cases} 1 & \text{if } 0 \leq t \leq 1 \\ \rho(n) - \displaystyle\int_n^t \frac{\rho(v-1)}{v} dv & \text{if } n \leq t \leq n+1 \end{cases}$$

$\rho(t)$ is thus an approximation of the probability that a u-bit number is $2^{u/t}$-smooth; Table 3 gives the numerical value of $\rho(t)$ (on a logarithmic scale) for $1 \leq t \leq 10$.

t	1	2	3	4	5	6	7	8	9	10
$-\log_2 \rho(t)$	0.0	1.7	4.4	7.7	11.5	15.6	20.1	24.9	29.9	35.1

Table 3. The value of Dickman's function for $1 \leq t \leq 10$.

One can use the following theorem [12] to estimate the average number of y values tried before such a factorization is obtained.

Theorem 2. *Let x be an integer and let $L_x[\beta] = \exp\left(\beta \cdot \sqrt{\log x \log\log x}\right)$. Let t be an integer randomly distributed between zero and x^γ for some $\gamma > 0$. Then for large x, the probability that all the prime factors of t are less than $L_x[\beta]$ is given by :*

$$L_x\left[-\frac{\gamma}{2\beta} + o(1)\right]$$

Using this theorem, an asymptotic analysis of Desmedt and Odlyzko's attack is given in [17]. The analysis yields a time complexity of:

$$L_x[\sqrt{2} + o(1)]$$

where x is a bound on $\mu(m)$. This complexity is sub-exponential in the size of the integers $\mu(m)$. Therefore, without any modification, Desmedt and Odlyzko's attack will be practical only if $\mu(m)$ is small.

D Optimizing Bernstein's Batch Size

We assume that for a single batch the algorithm runs in time:

$$\text{BatchTime}(n'_t, a, \ell) = c \cdot b' \cdot \log^2 b' \cdot \log\log b'$$

where c is a constant and:

$$b' = n'_t \cdot a + u \tag{7}$$

is the bit-length of the batch, and $u = \ell \cdot \log_2 \ell$ is the p_i-list's size in bits. The total running time is then:

$$\text{TotalTime}(n_t, a, \ell, n'_t) = \frac{n_t}{n'_t} \cdot c \cdot b' \cdot \log^2 b' \cdot \log\log b'$$

The running time of a single batch only depends on b'. Hence, as a first approximation one could select an n'_t equating the sizes of the t_i-list and the p_i-list; this yields $n'_t \cdot a = u$. A more accurate analysis (see below) reveals that TotalTime is minimized for a slightly larger n'_t value; more precisely for an n'_t such that:

$$n'_t \cdot a = \frac{u \log u}{2}$$

Using (7) this gives $b' = (u \log u)/2$ and a total running time of:

$$\text{TotalTime}(n_t, a, \ell) \simeq c \cdot n_t \cdot a \cdot \log^2 b' \cdot \log\log b'$$

We now proceed with the analysis of the optimal n'_t. For the sake of clarity we temporarily denote b' by b. Let $u = \ell \cdot \log_2 \ell$ and

$$n'_t = \frac{u}{a} \cdot \alpha$$

for some parameter α. We look for the optimal α. We have $b = u \cdot (\alpha + 1)$ and:

$$\text{TotalTime}(n_t, a, \ell, \alpha) = \frac{n_t \cdot a}{u \cdot \alpha} \cdot c \cdot b \cdot \log^2 b \cdot \log\log b$$

We neglect the $\log\log b$ term and consider the function:

$$f(u, \alpha) = \frac{b \cdot \log^2 b}{\alpha} \quad \text{where} \quad b = u \cdot (\alpha + 1)$$

Setting:

$$\frac{\partial f(u, \alpha)}{\partial \alpha} = 0$$

we get:

$$u \cdot (\log^2 b + 2 \log b) \cdot \alpha - b \log^2 b = 0 \quad \textit{i.e.} \quad (\log b + 2) \cdot \alpha = (\alpha + 1) \log b$$

and then:

$$2\alpha = \log b$$

which gives:

$$2\alpha = \log u + \log(\alpha + 1)$$

Neglecting the $\log(\alpha + 1)$ term, we finally get:

$$\alpha \simeq \frac{\log u}{2}$$

as the optimal α. This translates into running time as:

$$\text{TotalTime}(n_t, a, \ell) \simeq c \cdot n_t \cdot a \cdot \log^2 b \log\log b$$

where $b = (u \log u)/2$ and $u = \ell \cdot \log_2 \ell$.

E Large Prime Variant: Complexity Analysis

In this appendix we provide an accurate analysis of the large prime variant in the context of our attack.

Assume that we check our t_i-list for (B, B_2)-semi-smoothness (instead of B-smoothness) and detect η semi-smooth numbers. Amongst those, we expect to find $\eta\lambda$ numbers that are actually B-smooth, for some $\lambda \in [0,1]$ that can be expressed in terms of ρ and σ functions. If we further assume that the $\eta(1-\lambda)$ remaining numbers, which are semi-smooth but non-smooth, have their largest prime factors uniformly distributed amongst the h primes between B and B_2, we expect to find about $\eta^2(1-\lambda)^2/(2h)$ "collisions" between them, that is, about $\eta^2(1-\lambda)^2/(2h)$ pairs of numbers with the same largest prime factor.

Note that:

$$h \simeq \frac{B_2}{\log B_2} - B$$

Let ℓ be the number of primes less than B. The smooth numbers in the list yield a total of $\eta\lambda$ exponent vectors over the first ℓ primes, and each of the collisions between the remaining semi-smooth numbers yields such an additional exponent vector. Since we need (slightly more than) ℓ vectors to forge a signature, we should examine enough t_i's to find η semi-smooth numbers, where η satisfies:

$$\ell = \eta\lambda + \frac{\eta^2(1-\lambda)^2}{2h}$$

Solving for η, we get:

$$\eta = \frac{2\ell}{\lambda + \sqrt{2\ell \cdot (1-\lambda)^2/h + \lambda^2}}$$

The probability β that a random a-bit integer is semi-smooth with respect to B_2 and $B \simeq \ell \cdot \log \ell$ is:

$$\beta = \sigma\left(\frac{a \log 2}{\log(\ell \log \ell)}, \frac{a \log 2}{\log B_2}\right)$$

and if α denotes the probability that a random a-bit integer is B-smooth, we have:

$$\lambda = \frac{\alpha}{\beta} = \frac{\rho\left(\frac{a \log 2}{\log(\ell \log \ell)}\right)}{\sigma\left(\frac{a \log 2}{\log(\ell \log \ell)}, \frac{a \log 2}{\log B_2}\right)}$$

In this large prime variant, we only need to generate $n'_t = \eta/\beta$ numbers to find enough exponent vectors, as opposed to $n_t = \ell/\alpha$ previously. Therefore, the large prime variant improves upon simple smoothness by a factor of roughly:

$$\vartheta = \frac{n_t}{n'_t} = \frac{\ell/\alpha}{\eta/\beta} = \frac{1}{\lambda} \cdot \frac{\ell}{\eta} = \frac{1}{2}\left[1 + \sqrt{1 + \frac{2\ell}{h}\left(\frac{1}{\lambda} - 1\right)^2}\right] \geq 1 \tag{8}$$

ϑ is always greater than 1, and for the sizes we are interested in, say $100 \leq a \leq 200$, we find $\vartheta \simeq 1.5$ for the best choice of B, and $B_2 \gtrsim 7B$.[4] The reader is referred to Table 4 for precise figures.

[4] According to formula (8), ϑ increases until B_2 reaches $\simeq 7B$, and decreases slowly thereafter. This is actually *not* the case: finding a larger t_i population to be semi-smooth can only produce *more* collisions. The decrease suggested by formula (8) stems from the assumption that the largest prime factors of the t_i's are uniformly distributed amongst the h primes between B and B_2, which is only approximately true. The imprecision grows with h (a larger B_2 doesn't spread the largest prime factors more thinly). Choosing a very large B_2 is not advisable, however, because it produces considerable extra output (searching for collisions becomes cumbersome) with negligible returns in terms of actual collisions. In the practical attack, we selected $\ell = 2^{20}$ and $B_2 = 2^{27} \simeq 9B$.

a	128	144	160	176	192
Optimal $\log_2(\ell)$	19	20	21	23	24
Best ϑ	1.43	1.46	1.49	1.43	1.45

Table 4. Improvement factors due to the large prime variant.

F ISO/IEC 9796-2 Attack: Optimal $\{a, b\}$ Pairs

We provide in Table 5 the optimal $\{a, b\}$ pairs for for several RSA challenge moduli.

Challenge	RSA-704	RSA-768	RSA-896	RSA-1024	RSA-1536	RSA-2048
a	481	251	775	311	581	625
b	228	132	412	172	316	332

Table 5. $\{a, b\}$ values for several RSA challenge moduli.

G $\mu(x, 0)$ for RSA-2048 ISO/IEC 9796-2 Forgery

$$\begin{array}{rl}
\mu(x,0) = & \texttt{6a}\underline{\texttt{05b0daa253785bc8ab962c6047d6171eab4429160aa5defc7ff64bcc0a1de4}} \\
& \underline{\texttt{6b48a5c79a4e3394a66da695bb972de0be0bd94e40e8bad820b2b942aa8e71db}} \\
& \underline{\texttt{bca41cb3c2e7f2a68c88f4aa47a70a491199ef13b6a75b0650b22b4b2547f72e}} \\
& \underline{\texttt{f607c019702fdaa4f406a00cee511fcc5dd67dc2cd52c519976bb74971974b40}} \\
& \underline{\texttt{cf0e5459b5d18b7e15f338db671eb02f5ea32db8bc4a0bcf5cc896894fe1e738}} \\
& \underline{\texttt{6395f2f6f5c76ff7b6056ab5a5dfa84c95a117316615586bffb701d8716fd0bd}} \\
& \underline{\texttt{2712ada91a94d2d516246b3910d717cc9de96c48d2bc825994af77d0c283b93f}} \\
& \underline{\texttt{67f3c8256aeb45bab7037c}}\texttt{0000000000000000000000000000000000000000bc}_{16}
\end{array}$$

A chunk of $\mu(x, 0)$ was underlined for the ease of further reference.

H LLL Attack on EMV SDA-IPKD Encoding

H.1 The LLL Attack

Given a, N we must minimize the value of:

$$\left| b \cdot N - a \cdot X \cdot 2^{k_2} - a \cdot (\texttt{6A02}_{16} \cdot 2^{k_1} + Y \cdot 2^{k_3}) \right|$$

We show how this can be done using LLL. We write:

$$\begin{aligned} u &= a \cdot 2^{k_2} \\ v &= a \cdot (\texttt{6A02}_{16} \cdot 2^{k_1} + Y \cdot 2^{k_3}) \end{aligned}$$

where $N \simeq 2^k$, $X \simeq 2^{48}$, $a \simeq 2^{k_a}$, $u \simeq 2^{k-64+k_a}$ and $v \simeq 2^{k+k_a}$.

Hence we must minimize the absolute value of:

$$t = b \cdot N - x \cdot u - v$$

Consider the lattice of column vectors:

$$L = \begin{bmatrix} & & 2^{k-48} \\ & 2^{k-96} & \\ N & -u & -v \end{bmatrix}$$

As seen previously, heuristically, we can obtain $t \simeq 2^{k-48}$; therefore the coefficients in L are chosen so as to obtain a short vector of norm $\simeq 2^{k-48}$. More precisely, we look for a short column vector $\boldsymbol{c} \in L$ of the form:

$$\boldsymbol{c} = \begin{bmatrix} 2^{k-48} \\ x \cdot 2^{k-96} \\ b \cdot N - u \cdot x - v \end{bmatrix}$$

Theorem 3 (LLL). *Let L be a lattice spanned by $(u_1, \ldots, u_\omega)$. The* LLL *algorithm, given the vectors $(u_1, \ldots, u_\omega)$, finds in polynomial time a vector b_1 such that:*

$$\|b_1\| \leq 2^{(\omega-1)/4} \det(L)^{1/\omega}$$

Therefore, using LLL we can find a short vector of norm:

$$\|b_1\| \leq 2 \cdot (\det L)^{1/3} \leq 2 \cdot (2^{3k-144})^{1/3} \leq 2^{k-47}$$

Heuristically we hope that $b_1 = \boldsymbol{c}$, which allows solving for the values of b and X. The attack is heuristic but it works very well in practice, as shown in the next section.

H.2 Practical Value for EMV SDA-IPKD

Consider again the SDA-IPKD EMV format; we write:

$$\mu(X, N_{\text{I},1}, h) = \texttt{6A02}_{16} \cdot 2^{k_1} + X \cdot 2^{k_2} + Y \cdot 2^{k_3} + N_{\text{I},1} \cdot 2^{k_4} + h$$

where the constant Y is taken to be:

$$Y = \texttt{010203 0101 F8 01}_{16}$$

The first 3 bytes correspond to the CSN assigned by the CA (we took $\texttt{010203}_{16}$), $\texttt{0101}_{16}$ corresponds to the *hash algorithm indicator* and to the *public-key algorithm indicator*. $\texttt{F8}_{16} = 248$ is the issuer public-key length (in bytes) and $\texttt{01}_{16}$ is the length of the public exponent ($e = 3$).

Taking the RSA-2048 challenge for N, we have run the attack (Appendix H.1) for $k_a = 52$ and found the following values after 8,303,995 $\simeq 2^{23}$ iterations:

$$a = 4127135343129068 \qquad b = 2192055331476458 \qquad X = 66766242156276$$

which are such that $0 < X < 2^{48}$ and:

$$0 \leq b \cdot N - a \cdot \mu(X, 0, 0) < a \cdot 2^{k_3} \tag{9}$$

as required.

The computation took $\simeq 109$ minutes on a single 2GHz PC. Therefore, for $k_a = 36$ we expect that 2^{36} trials to yield a triple $\{a, b, X\}$ satisfying condition (9) such that $|a| \leq 2^{36}$, within a running time of $\simeq 109 \cdot 2^{36-20} = 4.3 \cdot 10^8$ minutes = 13 years on a single PC, or equivalently for US\$11,000 using the EC2 grid.

I EMV Signature Encoding Formats

EMV specifies the following encoding formats, based on the ISO/IEC 9796-2 standard. The new attack applies to modes preceded by the sign ♦ and does not apply to modes preceded by a ◊.

1. ♦ *Static Data Authentication, Issuer Public Key Data*
EMV SDA-IPKD §5.1, Table 2, page 41.
The signing entity is the CA. The signed message is the Issuer's public-key P_{I}.

$$m = \mathtt{02}_{16} \| X \| Y \| N_{\mathrm{I}} \| \mathtt{03}_{16}$$

While being operationally debatable we assume that X (the concatenation of the *Issuer's Identifier* (4 bytes) and the *Certificate Expiration Date* (2 bytes)) and N_{I} (the Issuer's modulus to be certified) can be both controlled by the attacker. Y (7 bytes) cannot be controlled by the adversary.

2. ◊ *Static Data Authentication, Static Application Data*
EMV SDA-SAD §5.1, Table 3, page 42.
The signing entity is the Issuer. The signed message is the Issuer's public-key P_{I}. As the first part of the message m is fixed, the attack does not apply.

3. ♦ *Offline Dynamic Data Authentication, Issuer Public-Key Data*
EMV ODDA-IPKD §6.1, Table 10, page 57.
The signing entity is the CA. The signed message is the Issuer's public-key P_{I}. The message format is identical to SDA-IPKD.

4. ♦ *Offline Dynamic Data Authentication,* ICC *Public-Key Data*
EMV SDA-ICC-PKD §6.1, Table 11, page 58.
The signing entity is the Issuer. The signed message is the Card's public-key P_{I}.

$$m = \mathtt{04}_{16} \| X \| Y \| N_{\mathrm{ICC}} \| \mathtt{03}_{16} \| \mathrm{DATA}$$

While being operationally debatable we assume that X (12 bytes) and N_{ICC} (the Card's modulus to be certified) can be both controlled by the attacker. Y (7 bytes) cannot be controlled by the adversary. Here DATA is static data to be authenticated, as specified in Section 10.3, Book 3, EMV specifications; DATA can only appear in the non-recoverable part of the message in the ISO/IEC 9796-2 standard.

5. ◊ *Offline Dynamic Data Authentication, Dynamic Application Data*
EMV ODDA-DAD1 §6.5, Table 15, page 67.
The signing entity is the Card. As the first part of the message m is fixed ($\mathtt{BB}_{16}$ padding), the attack does not apply.

6. ◊ *Offline Dynamic Data Authentication, Dynamic Application Data*
EMV ODDA-DAD2 §6.6, Table 18, page 73.
The signing entity is the Card. As the first part of the message m is fixed[5], the attack does not apply.

[5] Here $m = \mathtt{0501}_{16} \| Y \| X$ where Y is a $32 + 1 = 33$ bytes string that cannot be controlled by the adversary (32 leftmost bytes of ICC Dynamic Data). X can be controlled by the adversary.

7. ♦ *Personal Identification Number Encipherment,* ICC PIN *Encipherment Public Key Data*
EMV ICC PIN §7.1, Table 23, page 83.
The signing entity is the Issuer.

$$m = \texttt{04}_{\texttt{16}} \| X \| Y \| N_{\text{ICC}} \| \texttt{03}_{\texttt{16}}$$

While being operationally debatable we assume that X (12 bytes) and N_{ICC} (the Card's modulus to be certified) can be both controlled by the attacker. Y (7 bytes) cannot be controlled by the adversary.

Commercial impact. It is very fortunate that ODDA-DAD1 and ODDA-DAD2 do not lend themselves to the new attack. Indeed, in the ODDA-DAD modes the signing device is the payment card, which is supposedly in the opponent's hands. In addition, the signing capacity of EMV cards is limited by a ratification counter restricting the number of signatures performed by the card during its lifetime.

J Evidencing Forgery - Experimental Results

Let:

$$\texttt{HashList}(m_i) = \text{SHA-1}(v_{i,1}, \dots, v_{i,\ell}) \quad \text{where} \quad \mu(m_i) = \prod_{j=1}^{\ell} p_j^{v_{i,j}}$$

To evidence forgery it suffices to exhibit a handful of m_i such that $\texttt{HashList}(m_i) < 2^{160-w}$ for some bound w. Given that $v_{i,1}, \dots, v_{i,\ell}$ cannot be determined before actually factoring $\mu(m_i)$, it is easy to estimate[6] the probability $p(a, b)$ of discovering a solutions of the inequality

$$\texttt{HashList}(x) < 2^{160-w}$$

before trying b successful factorizations. The number of such solutions follows a Poisson distribution $\lambda^k e^{-\lambda}/k!$ of mean $\lambda = 2^{-w}b$. Hence:

$$1 - p(a, b) \simeq \frac{\Gamma(a, 2^{-w}b)}{(a-1)!}$$

where:

$$\Gamma(x, y) = \int_y^{+\infty} t^{x-1} e^{-t}\, dt$$

is the incomplete Gamma function.

Let:

$$m_\bullet = \psi \| \texttt{a730b8f5b3bfaf619ad3cc18} \| \Delta_\bullet$$

Here ψ represents the digits underlined in Appendix G, the bullet symbol ($\bullet$) stands for either a single digit i or a couple of digits of the form $\{i, 1\}$ or $\{i, 2\}$ and:

[6] Under proper randomness assumptions.

$$\begin{array}{ll}
\Delta_{1,1} = \texttt{0056000f88dd8570}_{16} & \Delta_{1,2} = \texttt{0080000b9c26d0eb}_{16} \\
\Delta_{2,1} = \texttt{00530013cf47845a}_{16} & \Delta_{2,2} = \texttt{0053000fdc84e93f}_{16} \\
\Delta_{3,1} = \texttt{001000102c6c2b50}_{16} & \Delta_{3,2} = \texttt{01d2000959b3fe67}_{16} \\
\Delta_{4,1} = \texttt{01720003b13b774a}_{16} & \Delta_{4,2} = \texttt{00b90005f7322db0}_{16} \\
\Delta_{5,1} = \texttt{016c00023dfe3e51}_{16} & \Delta_{5,2} = \texttt{007600120351e250}_{16} \\
\Delta_{6,1} = \texttt{00c0000e367388fc}_{16} & \Delta_{6,2} = \texttt{01cf000978d5b919}_{16} \\
\Delta_{7,1} = \texttt{00800003d69f1d8f}_{16} & \Delta_{7,2} = \texttt{0190000171baf33f}_{16}
\end{array}$$

$$\begin{array}{lll}
\Delta_{8} = \texttt{00730011166d31ca}_{16} & \Delta_{9} = \texttt{008500069b2f2c24}_{16} & \Delta_{10} = \texttt{0130000976d3e847}_{16} \\
\Delta_{11} = \texttt{016a0006a13b9147}_{16} & \Delta_{12} = \texttt{016d0004e0c4a393}_{16} & \Delta_{13} = \texttt{0005000ce3085400}_{16} \\
\Delta_{14} = \texttt{0048000a6b9e765a}_{16} & \Delta_{15} = \texttt{0051000cf1f5f33d}_{16} & \Delta_{16} = \texttt{005500124f563dab}_{16}
\end{array}$$

The $\{m_{i,1}, m_{i,2}\}$'s represent semismooth message-pairs whereas m_i's stand for smooth messages. Let N denote the RSA-2048 challenge module, for which $\{a, b\} = \{625, 332\}$, and define $(v_{i,1}, \ldots, v_{i,\ell})$ as follows:

$$\prod_{j=1}^{\ell} p_j^{v_{i,j}} = \begin{cases} bN - a\mu(m_i) & \text{if } \bullet \text{ is a single digit } i \\ \frac{bN - a\mu(m_{i,1})}{bN - a\mu(m_{i,2})} & \text{if } \bullet \text{ is a couple of digits of the form } \{i,1\} \text{ or } \{i,2\} \end{cases}$$

Form an ASCII string s_i of decimal digits separated by comas representing the indices of the nonzero coordinates of $(v_{i,1} \bmod 2, \ldots, v_{i,\ell} \bmod 2)$:

s_1 = `"1,2,12,16,21,50,143,150,1188,5610,23440,45965,73835,211017,249051,665273,719577,757330,1044994"`

s_2 = `"1,5,6,8,16,40,68,524,5831,7882,7935,40769,53518,248737,357717,410827,490213,500776,722037"`

s_3 = `"6,10,12,16,19,21,40,46,92,110,189,241,779,1031,1681,2311,3495,3787,4183,8262,123354,162437,846636"`

s_4 = `"5,6,8,9,10,44,186,329,371,538,1081,2927,7396,12857,63132,126069,166154,521440,560769"`

s_5 = `"4,5,13,48,83,161,218,284,370,2555,3667,11247,14293,164010,206877,239786,306991,605844,800833"`

s_6 = `"1,2,6,7,25,886,1831,4144,13634,27788,30584,38351,47012,122884,170598,361555,573756,743547"`

s_7 = `"6,25,62,81,119,267,330,3275,3567,4351,9761,19676,29917,60169,73891,131694,589244,876506"`

s_8 = `"2,7,17,32,56,379,579,7145,17165,18312,128999,605951"`

s_9 = `"1,31,80,125,5680,7782,19807,28271,37153,600595,856133"`

s_{10} = `"1,9,17,791,3062,3745,7809,51989,272289,290253,413869"`

s_{11} = `"1,6,7,9,14,429,1076,19665,20226,33554,43469,59463,179807"`

s_{12} = `"1,5,8,93,266,46588,69875,95818,225171,462306,567353"`

s_{13} = `"13,32,1287,2620,5387,7457,98930,165992,434883,447877"`

s_{14} = `"4,11,19,152,1321,3752,14469,30192,75013,182786,1028476"`

s_{15} = `"1,6,21,431,3562,15278,42172,117602,199652,514092,721130"`

s_{16} = `"34,342,584,621,636,1588,2198,2503,17225,53374,145619"`

and define: `HashList`$(m_i) = $ SHA-1(s_i).

We have:

message(s)	$\texttt{HashList}(m_i)$ or $\texttt{HashList}(\{m_{i,1}, m_{i,2}\})$
$\{m_{1,1}, m_{1,2}\}$	$\texttt{0000803c0b8d1e00a35d52fe2dac52c3c3ad99dd}_{16}$
$\{m_{2,1}, m_{2,2}\}$	$\texttt{00005d4a8f8e2fe787d4771dfd18ed763ee0ed8d}_{16}$
$\{m_{3,1}, m_{3,2}\}$	$\texttt{0000567540f6e3e377c708ba423f70ab7fcbae8d}_{16}$
$\{m_{4,1}, m_{4,2}\}$	$\texttt{0000a9e0b8a80f8db3a98af7b231d362d2e425ed}_{16}$
$\{m_{5,1}, m_{5,2}\}$	$\texttt{0000feb689f4b32fe20de83a0a25fd0e3288484b}_{16}$
$\{m_{6,1}, m_{6,2}\}$	$\texttt{0000775083325da0234d34b652d23ac500631a97}_{16}$
$\{m_{7,1}, m_{7,2}\}$	$\texttt{0000f5a3e13aa3569bde1755ec4aa6358ccb3511}_{16}$
m_8	$\texttt{0000b864ed46151c824ef412fb6d2de4df5d8c74}_{16}$
m_9	$\texttt{0000466f85211cd79a73f8b6afa6a7912d8ac13f}_{16}$
m_{10}	$\texttt{0000c339f0ea8b8a108b25d16a54d3c8203c30c8}_{16}$
m_{11}	$\texttt{00007f89d2d05f1e0b08730a18466bcb454c4176}_{16}$
m_{12}	$\texttt{00000cf8807e716f305bc0ea47e7664d3bb4917e}_{16}$
m_{13}	$\texttt{0000ef3c35c43074e8fea95b7d4a496430686acf}_{16}$
m_{14}	$\texttt{0000d64a56a66619879b313ffed1a200d64223ed}_{16}$
m_{15}	$\texttt{0000e5b867f4d70f61f586ca3df6b5893370e847}_{16}$
m_{16}	$\texttt{0000356512c0c73ac4235447793af436247d02c6}_{16}$

The probability to obtain $a = 16$ digests smaller than $2^{160-w} = 2^{160-16} = 2^{144}$ without being in actual possession of a 2^{19}-column forgery matrix is smaller than 1%. 650,000 columns are necessary to reach 5%. Hence, one can conclude with 95% confidence, that the authors possess at least 63% of the digests allowing to forge ISO/IEC 9796-2 signatures. The evidence (set of Δ_i) is indeed very compact.

We do not know how to generate compact evidence in support of the algebraic phase. However, since the algebraic phase is much easier, it is safe to assume that he who can do more, can do less. In any case, a compact *proof*, rather than statistical evidence, would of course be desirable.

K Fewer Queries

The number of signatures *actually used* by the forger is not τ but the number of nonzero β_i values in the formula:

$$\mu(m_\tau) = \left(\prod_{j=1}^{\ell} p_j^{\gamma_j} \right)^e \cdot \prod_{i=1}^{\tau-1} \mu(m_i)^{\beta_i}$$

Assuming that $(\beta_1, \ldots, \beta_{\tau-1})$ is a random vector of $\mathbb{Z}_e^{\tau-1}$ only $\tau(e-1)/e$ of the signatures will be actually used to compute the forgery. The gain is significant when e is a very small exponent (*e.g.* 2 or 3). However, one can try to generate more than τ candidates but select the subset of signatures minimizing the number of nonzero β_i values. Such a sparse β-vector may allow to reduce the number of queries and defeat *ratification counters* meant to restrict the number of authorized signature queries.

In essence, we are looking at a random $[\ell, k]$ code: a kernel vector has ℓ components which, for $e = 2$, can be regarded as a set of independent unbiased Bernoulli variables. The probability that such a vector has weight less than $w = \sum_{i=1}^{\tau-1} \beta_i$ is thus:

$$\sum_{j=1}^{w} \binom{\ell}{j} 2^{-\ell} \simeq \frac{1}{2} \left(1 + \operatorname{erf} \left(\frac{w - \ell/2}{\sqrt{\ell/2}} \right) \right)$$

We have 2^k such vectors in the kernel, hence the probability that at least one of them has a Hamming weight smaller than w is surely bounded from above by:

$$2^k \times \frac{1}{2}\left(1+\operatorname{erf}\left(\frac{w-\ell/2}{\sqrt{\ell/2}}\right)\right) = 2^{k-1}\left(1+\operatorname{erf}\left(\frac{w-\ell/2}{\sqrt{\ell/2}}\right)\right)$$

Let c denote the density bias of w *i.e.*, $w = (1/2 - c)\ell$. The previous bound becomes:

$$p(c) = 2^{k-1}\left(1+\operatorname{erf}\left(-c\sqrt{2\ell}\right)\right) = 2^{k-1}\left(1-\operatorname{erf}\left(c\sqrt{2\ell}\right)\right) = 2^{k-1}\operatorname{erfc}(c\sqrt{2\ell}) \underset{\ell\to+\infty}{\sim} \frac{2^{k-1}\exp(-2\ell c^2)}{c\sqrt{2\pi\ell}}$$

For $\ell = 2^{20}$, even if we take k as large as 2^{10} (the largest subspace dimension considered tractable, even in much smaller ambient spaces), we get $p(1/50) \simeq 10^{-58}$, so the probability that there exists a kernel vector of weight $w < 500{,}000$ is negligible. In addition, even if such a vector existed, techniques for *actually computing it*, *e.g.* [10], seem to lag far behind the dimensions we deal with.

It follows that a better strategy to diminish w is to simply decrease ℓ. The expected payoff might not be that bad: If the attacker is limited to, say, 2^{16} signatures, then he can pick $\ell = 2^{17}$, and for 196-bit numbers (204 bits minus 8 bits given by exhaustive search), the attack becomes about 15 times slower than the optimal choice, $\ell = 2^{24}$ (note as well that more exhaustive search becomes possible in that case). That's slow, but perhaps not excruciatingly so.

L Expected Number of Queries

A further question that arises when studying the problem of minimizing the number of signature queries in the attack is that of evaluating the expected Hamming weight of the nullspace vectors returned by the linear algebra step, with or without the large prime variant. The problem is nontrivial and still open. For simplicity we only consider the GF(2) case ($e = 2$) the following, but the problem is not fundamentally different for other exponents.

Some primes can never appear as components of nullspace vectors. That is the case, in particular, when the corresponding row of the matrix contains only one nonzero element (*i.e.* if the prime divides only one of our smooth or semismooth numbers to an odd power): indeed, the column containing that nonzero element is clearly linearly independent from all other columns. Hence, we may as well remove that row and column from the matrix, and do so for every similar pair of rows and columns.

Additional rows of weight 1 may turn up after those removals, so we need to repeat the process recursively. A fix-point is reached (*i.e.* the algorithm stops) when there is no singleton row left in the reduced matrix.

Let ℓ_0 denote the total number of rows in this reduced matrix. It seems reasonable to conjecture that (in the case of a square matrix at least) the expected weight of nullspace vectors is of the order of $\ell_0/2$. In other words, the primes that *can* appear as components of nullspace vectors are likely to do so randomly. The question then reduces to evaluating ℓ_0.

All this prompts us to estimate how many rows in the original matrix contain exactly one nonzero entry, or more generally exactly n nonzero entries for small n. We expected this variable to follow a Poisson distribution, or perhaps a sum of Poisson distributions, but experiments appear to challenge this expectation: the empirical distribution does not have exponential decay but a fat tail, as evidenced by the following plots. Finding a simple description of this distribution seems like a rather difficult open problem in itself.

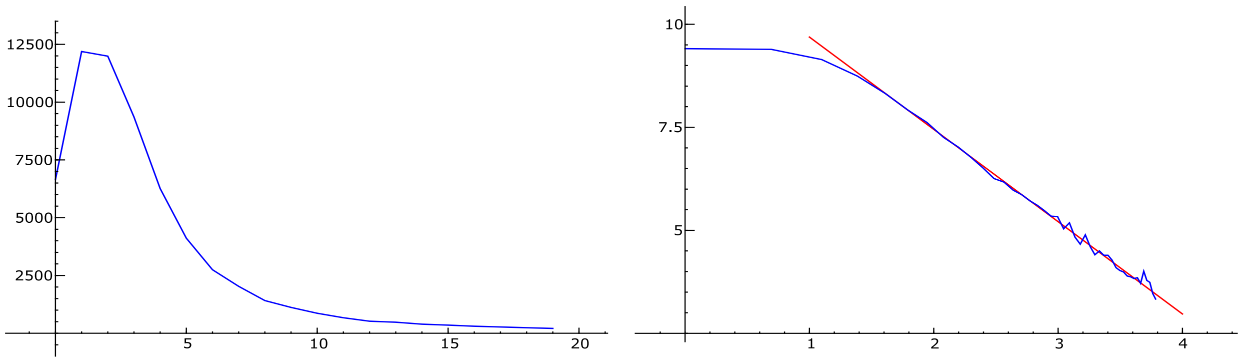

Fig. 1. Distribution of row weights for a smooth matrix of size 2^{16}. Linear scale (left) and log-log scale (right).

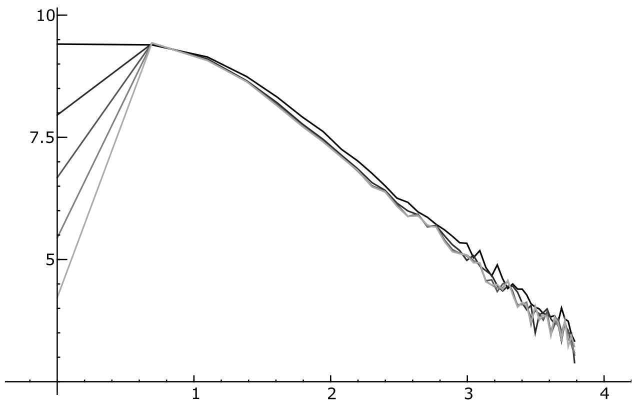

Fig. 2. Distribution of row weights (log-log scale) for a smooth matrix of size 2^{16} and the first four iterations of the reduction step applied to it. Lighter curves correspond to further iterations.

M Faster Hashing Options

Evaluating the practical impact of the attacks described in this paper requires estimating the speed at which t_i inputs can be fed into the smoothness detection phase. Creating a t_i candidate is in essence as fast as hashing a single SHA-1 message block.

A standard PC (single AMD core in a quad-core Opteron 8356 running OpenSSL's SHA-1 implementation compiled in 64-bit mode) can perform $2^{23.3}$ SHA-1 compression function evaluations per second. We refer the reader to [5] for a systematic performance study of diverse hash-functions on a variety of x86 and x86-64 platforms.

FPGA **clusters** : COPACOBANA [46] is an optimized FPGA machine developed for cryptanalytic purposes. The device consists of 120 Xilinx Spartan 3 FPGAs, clocked at 40MHz. A SHA-1 compression function evaluation on COPACOBANA claims 40 clock cycles [38] (*i.e.* 2^{30} digests/FPGA/second). In other words, the machine achieves an overall hashing throughput of $\simeq 2^{37}$ digests/second. A device costs $\simeq$ US\$12,000.

Video cards start being used for cryptographic calculations. Dedicated programming languages for NVIDIA's CUDA [44] allow to offload computationally intensive tasks onto graphic cards. Pyrit project[7] data allows to estimate that $\simeq 2^{27}$ digests/second are achievable on a high-end NVIDIA GTX 280 graphics card (the GTX 280 card, costing US\$280 offers the best price/performance as we write these lines). The high-end NVIDIA Tesla machine (US\$10,000) operated by the SAGE project [50] is faster but features a worse price/performance *ratio* than an equivalent-cost cluster of NVIDIA GTX 280 cards.

PlayStation 3s: The Cell Broadband Engine provides excellent parallel processing. Blade cell clusters are one option. A much cheaper option consists in using Sony PlayStations (3s model, US\$400/unit). A cluster of these devices was successfully used to find chosen-prefix MD5 collisions [52]. A single PlayStation 3s is capable of hashing (SHA-1) at a throughput of 45Gb/second, *i.e.* $\simeq 2^{26}$ digests/second [45].

Other hashing solutions exist: the VIA C3 and various other RISC CPUs (*e.g.* UltraSparc T2 or various ARM processors) contain a cryptographic co-processor that can efficiently used to hash. We did not investigate these options in further detail.

[7] Pyrit – Advances in attacking WPA-PSK: `http://code.google.com/p/pyrit`